The GNU Linear Programming Kit, Part 3: Advanced problems and elegant solutions

Maximizing the profitability of perfume and building a better basketball team

Rodrigo Ceron (rceron@br.ibm.com), Staff Software Engineer, IBM, Software Group

Summary: The GNU Linear Programming Kit (GLPK) is a powerful, proven tool for solving numeric problems with multiple constraints. This article, the third in a three-part series, uses GLPK and the `glpsol` client utility with the GNU MathProg language to solve a perfume production problem and a basketball lineup problem.

Date: 14 Nov 2006
Level: Intermediate
Activity: 4654 views
Comments:

This article is the third in a three-part series on using the GNU Linear Programming Kit. For an introduction to GLPK, read the first installment in the series, "The GNU Linear Programming Kit, Part 1: Introduction to linear optimization."

[All company and product names given in the example problems are fictional. -Ed.]

Brute production process

This problem comes from *Operations Research: Applications and Algorithms, 4th Edition*, by Wayne L. Winston (Thomson, 2004); see Resources below for a link.

> Rylon Corporation manufactures Brute and Chanelle perfumes. The raw material needed to manufacture each type of perfume can be purchased for $3 per pound. Processing one pound of raw material requires 1 hour of laboratory time. Each pound of processed raw material yields 3 ounces of Regular Brute Perfume and 4 ounces of Regular Chanelle Perfume. Regular Brute can be sold for $7/ounce and Regular Chanelle for $6/ounce. Rylon also has the option of further processing Regular Brute and Regular Chanelle to produce Luxury Brute, sold at $18/ounce, and Luxury Chanelle, sold at $14/ounce. Each ounce of Regular Brute processed further requires an additional 3 hours of lab time and $4 processing cost and yields 1 ounce of Luxury Brute. Each ounce of regular Chanelle processed further requires an additional 2 hours of lab time and $4 processing cost and yields 1 ounce of Luxury Chanelle. Each year, Rylon has 6,000 hours of lab time available and can purchase up to 4,000 pounds of raw material. Maximize Rylon's profit.

To summarize the important information about the problem:

- Rylon Corporation manufactures Brute and Chanelle perfumes.

- Raw material costs $3 per lb.
- Processing 1 lb. of raw material takes 1 hr. of laboratory time.
- Each lb. of raw material yields 3 oz. of Regular Brute Perfume and 4 oz. of Regular Chanelle Perfume.
- Regular Brute sells for $7/oz. and Regular Chanelle for $6/oz.
- Reprocessing yields Luxury Brute, sold at $18/oz., and Luxury Chanelle, sold at $14/oz.
- Each oz. of Regular Brute processed further costs an additional 3 hrs. of lab time and $4 in processing cost and yields 1 oz. of Luxury Brute.
- Each oz. of regular Chanelle processed further costs an additional 2 hrs. of lab time and $4 of processing cost and yields 1 oz. of Luxury Chanelle.
- Yearly constraints: 6,000 hrs. of lab time available, and Rylon can purchase up to 4,000 lbs. of raw material.

Before modeling this problem, let's look at the transformation from raw materials to end products:

Figure 1. Overview of Rylon production

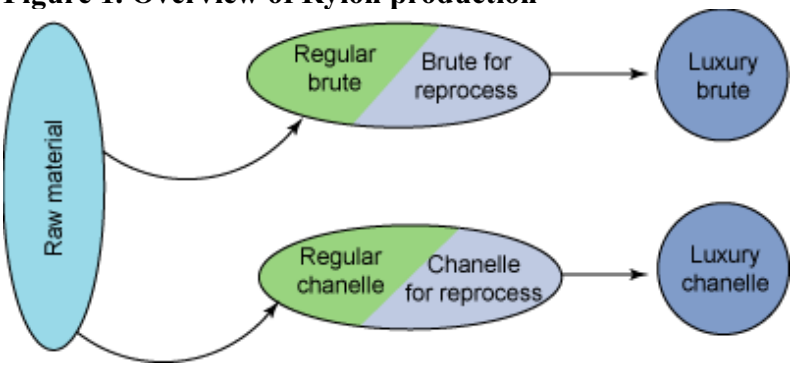


Figure 1 shows that all the raw material is transformed into intermediate products (represented by the intermediate balloons in the figure). Take the upper intermediate balloon as an example representing the Regular Brute product. Some of the Regular Brute may be reprocessed to generate the luxury Brute, so the balloon is split in two: the Regular Brute that will be sold (green) and the Regular Brute that will be reprocessed to make Luxury Brute (blue). The same applies to the Chanelle. Note that only the blue part of those two intermediate balloons (and all of the blue part) will be reprocessed.

Modeling the Rylon problem

The decision variables for this particular problem need to cover all the perfumes and materials:

- x_1: ounces of Regular Brute sold annually
- x_2: ounces of Luxury Brute sold annually
- x_3: ounces of Regular Chanelle sold annually
- x_4: ounces of Luxury Chanelle sold annually
- x_5: pounds of raw material purchased annually

The objective function can be written to maximize the profit in terms of these decision variables:

$$z = 7x_1 + (18 - 4)x_2 + 6x_3 + (14 - 4)x_4 - 3x_5$$

The trivial constraints from the problem are related to the laboratory hours available to process the

materials. Rylon needs time to process the amount of raw material it buys, which is given by x_5. Time is also needed to reprocess regular Brute and Chanelle into the luxury Brute and Chanelle, so the total time will depend on x_2 and x_4 in addition to x_5:

$$3x_2 + 2x_4 + x_5 \leq 6000 \ (lab\ hours)$$

The maximum amount of raw material Rylon can buy is also a constraint:

$$x_5 \leq 4000 \ (raw\ \ material)$$

As always, the sign constraints are important:

$$x_i \geq 0, \ \forall i \in \{1,..,5\}$$

If this model is converted to GNU MathProg and solved by `glpsol`, the solver will report that the problem is unbounded: the more Rylon produces, the bigger its profit is. Hey, this sounds like a good business if you want to move to the Caribbean Islands, but common sense says it can't happen. So, where's the flaw?

The raw material needs to be bound to the Regular Brute and Chanelle, and those two need to be bound to their Luxury versions. How can two variables be bound? The conversion of raw material to Chanelle and Brute is not 1 pound to one ounce, as you can see in the problem description. Each pound of raw material generates three ounces of regular Brute and four ounces of regular Chanelle (seven ounces of perfume in total). Think of this transformation as a black box. If you put one pound of raw material into the black box, you'll get exactly 3 ounces of regular Brute *and* four ounces of regular Chanelle out of the box. If you have processed `N` pounds of raw material, the upper intermediate balloon in Figure 1 *must* be `3N`. This is Regular plus Luxury Brute.

Remember that the transformation of one ounce of Regular perfume yields one ounce of Luxury perfume for both Brute and Chanelle. In addition, the lower intermediate balloon for Regular and Luxury Chanelle is `4N` ounces total. Rylon cannot, say, use `N` pounds of raw material to create `2N` ounces of Regular plus Luxury Brute and `5N` of Regular plus Luxury Chanelle. The 1-to-3 Brute transformation and 1-to-4 Chanelle transformation must both hold, regardless of the Regular-to-Luxury ratios within the Brute and Chanelle production yields.

The amount of perfume produced *must* have been generated from the raw material. If the equilibrium is not enforced, the results will be wrong! Hence the following constraint:

$$\frac{x_1 + x_2}{3} = x_5 \ (mass\ conservation\ for\ Brute)$$

This equation says that all the Brute produced (which is the Regular Brute for sale plus the Regular Brute for reprocessing) divided by 3 must equal one unit of raw material. Does that sound correct? The problem says that processing one unit of raw material (one pound) generates 3 units (3 ounces) of Regular Brute, and we know that 1 unit of Regular Brute may be transformed to 1 ounce of its Luxury line. Note here again that all the Regular and Luxury units (ounces) of Brute need to be exactly 3 times greater than the units of raw material processed. So, this constraint seems to be okay. Because GLPK requires that all decision variables be on the same side of the inequality, we write the previous equation as follows:

$$x_1 + x_2 - 3x_5 = 0$$

Similarly, we have that:

$$\frac{x_3 + x_4}{4} = x_5 \ (mass\ conservation\ for\ Chanelle)$$

Does that seem right? One unit of raw material generates 4 ounces of Chanelle (which is Regular plus Luxury), so this seems okay too. For GLPK, the equation is written with all the decision variables on the left:

$$x_3 + x_4 - 4x_5 = 0 \ (mass\ conservation\ for\ Chanelle)$$

Now, the problem is complete. Let's write a GNU MathProg program for it.

GNU MathProg solution for Rylon's problem

The following code for `glpsol` solves the Rylon problem. (The line numbers are not part of the code itself. I've added them to make it easier to reference the code later.)

Listing 1. Rylon production process solution

```
 1      #
 2      # Rylon production proccess
 3      #
 4      # This finds the optimal solution for maximizing Rylon's profit
 5      #
 6      /* sets */
 7      set PROD;
 8
 9      /* parameters */
10      param Rev{i in PROD};
11      param Cost{i in PROD};
12      param Labh{i in PROD};
13
14      var x{i in PROD} >= 0; /*x1: No. of oz of Regular Brute
15                               x2: No. of oz of Luxury Brute
16                               x3: No. of oz of Regular Chanelle
17                               x4: No. of oz of Luxury Chanelle
18                               x5: No. of lbs of raw material */
19
20      maximize z: sum{i in PROD} (Rev[i]*x[i] - Cost[i]*x[i]);
21
22      /* Constraints */
23      s.t. raw:   x['raw'] <= 4000;
24      s.t. time:  sum{i in PROD} Labh[i]*x[i] <= 6000;
25      s.t. mass1: x['rb'] + x['lb'] - 3*x['raw'] = 0;
26      s.t. mass2: x['rc'] + x['lc'] - 4*x['raw'] = 0;
27
28      data;
29      set PROD:= rb lb rc lc raw;
30
31      param Rev:=
32      rb      7
33      lb      18
```

```
34      rc      6
35      lc      14
36      raw     0;
37
38      param Labh:=
39      rb      0
40      lb      3
41      rc      0
42      lc      2
43      raw     1;
44
45      param Cost:=
46      rb      0
47      lb      4
48      rc      0
49      lc      4
50      raw     3;
51
52      end;
```

You should recognize all the declarations of the previous code, but I'll review them quickly for the sake of completeness.

Line 7 declares a set named `PROD`, whose elements are the Brute and Chanelle products. Line 29 defines the products. The four abbreviations stand for *Regular Brute*, *Luxury Brute*, *Regular Chanelle*, and *Luxury Chanelle*, and the last element is the raw material. Raw material is listed as a product so that all the decision variables can be in one set.

Lines 10, 11, and 12 declare the parameters: revenue, cost, and lab hours. The parameters are defined on lines 31 through 50. The lab hours say how long it takes to process a pound of raw material and to reprocess the Regular Brute and Chanelle into their Luxury versions (per ounce). The cost consists of the cost of the raw materials per pound and the cost of reprocessing the Regular Brute and Chanelle per ounce.

Line 14 defines the decision variables, a five-element array on the `PROD` set.

Line 20 defines the objective function, which is just Rylon's profit (revenue - costs).

Finally, lines 23 through 26 declare the problem's constraints, discussed above.

Listing 2 shows the output:

Listing 2. The Rylon report from glpsol

```
Problem:    brute
Rows:       5
Columns:    5
Non-zeros:  15
Status:     OPTIMAL
Objective:  z = 172666.6667 (MAXimum)

   No.   Row name   St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------- ------------- ------------- -------------
     1 z            B         172667
     2 raw          NU          4000                        4000       39.6667
     3 time         NU          6000                        6000       2.33333
     4 mass1        NS             0            -0             =             7
     5 mass2        NS             0            -0             =             6

   No. Column name  St   Activity     Lower bound   Upper bound    Marginal
```

```
------ ------------ -- ------------ ------------ ------------ -------------
     1 x[rb]        B        11333.3            0
     2 x[lb]        B        666.667            0
     3 x[rc]        B          16000            0
     4 x[lc]        NL             0            0                    -0.666667
     5 x[raw]       B           4000            0

Karush-Kuhn-Tucker optimality conditions:

KKT.PE: max.abs.err. = 1.46e-11 on row 1
        max.rel.err. = 8.43e-17 on row 1
        High quality

KKT.PB: max.abs.err. = 0.00e+00 on row 0
        max.rel.err. = 0.00e+00 on row 0
        High quality

KKT.DE: max.abs.err. = 8.88e-16 on column 2
        max.rel.err. = 5.92e-17 on column 2
        High quality

KKT.DB: max.abs.err. = 0.00e+00 on row 0
        max.rel.err. = 0.00e+00 on row 0
        High quality

End of output
```

The first section shows that the solution is optimal and that the objective function equals approximately 172667. The third section has the values of each decision variable. The second section shows the constraints. Note that the mass conservation constraints always have an = sign in one of the bounds columns. Their marginal values can't be analyzed, because it makes no sense to have a marginal value with equality constraints. Do you agree? Send me a note if you have a different opinion.

Let's make a brief, common-sense analysis here. Rylon has processed 4,000 pounds of raw material. According to the mass conservation (the black box that transforms raw material into Chanelle and Brute), we need to enforce that `N` pounds of raw material produce `3N` ounces of Brute (Regular plus Luxury) *and* also `4N` ounces of Chanelle (Regular plus Luxury). There are 16,000 ounces of Regular Chanelle and no ounces of Luxury Chanelle. So, we have the 1 (4,000 pounds) to 4 (16,000 ounces) proportion for Chanelle. Analogously, we also have the 1 (4,000 pounds) to 3 (11,333.333 ounces plus 666.667 ounces) proportion for Brute. Good, we are in the real world.

Without the mass conservation constraints, `glpsol` prints this error to `stdout` like so:

Listing 3. Errors without mass conservation

```
Reading model section from brute-production2.mod...
Reading data section from brute-production2.mod...
61 lines were read
Generating z...
Generating raw...
Generating time...
Model has been successfully generated
lpx_simplex: original LP has 3 rows, 5 columns, 9 non-zeros
PROBLEM HAS NO DUAL FEASIBLE SOLUTION
If you need actual output for non-optimal solution, use --nopresol
Time used:   0.0 secs
Memory used: 0.1M (149646 bytes)
lpx_print_sol: writing LP problem solution to `brute-production2.sol'...
```

It says that the problem has no dual feasible solution, but what does that mean? Every problem has a

dual equivalent. For example, a maximization problem can be rewritten as a minimization problem. The maximization problem is then called the *primal* problem, and the minimization is the dual problem. If the primal were a minimization problem, then the dual would be a maximization problem. *Dual* really means secondary or alternate, and *primal* means primary, or main, in plain English.

A primal problem that has no solution has an unbounded dual problem. An unbounded primal problem has a dual problem that's not feasible! Because `glpsol` said the dual problem is not feasible, the primal problem is unbounded (the Caribbean Islands situation I mentioned above).

Multiple solution problem

This section of the article describes a simple problem with multiple optimal solutions. All the problems I've presented in this series so far had only one optimal solution. A problem with multiple solutions has more than one point in its feasible region that either maximize or minimize a problem. The objective function for these critical points are the same, regardless. Here's an example.

Modeling a multiple solution problem

Maximize the following objective function:

$$z = x_1 + x_2$$

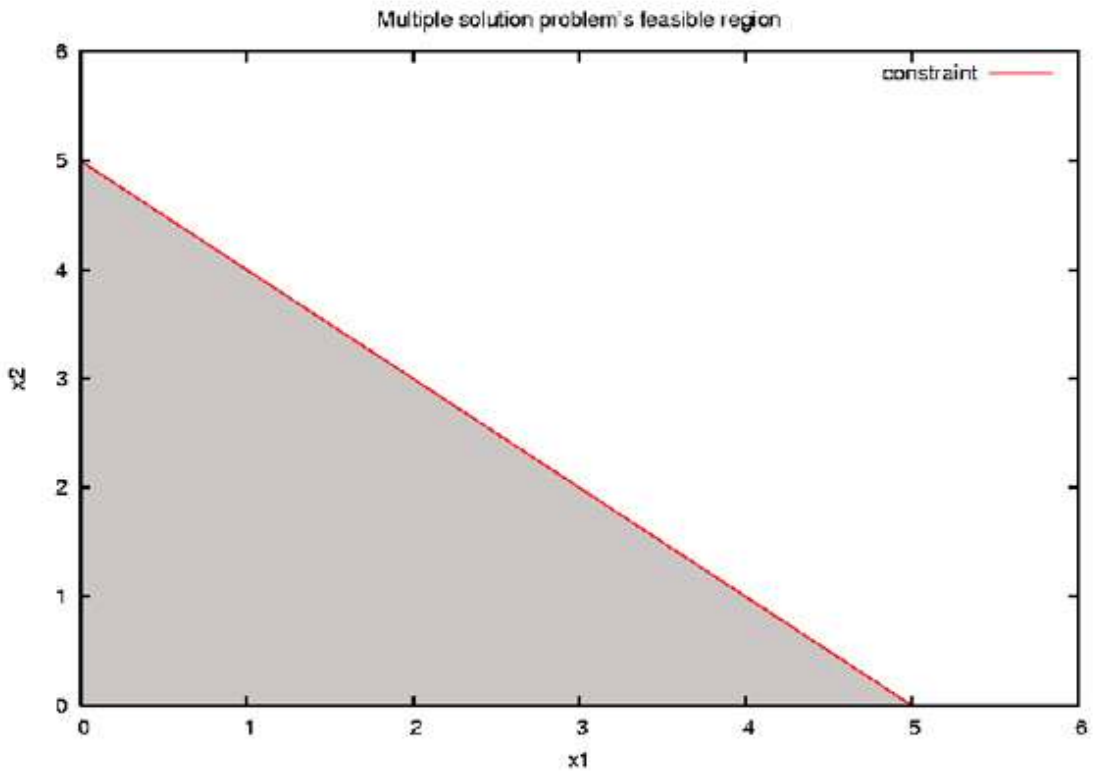
subject to the following two constraints:

$$x_1 + x_2 = 5$$

$$x_i \geq 0, \ \forall i \in \ \{1, 2\}$$

The feasible region for this particular problem is given by the gray area of Figure 2.

Figure 2. Feasible region of the multiple solution problem



Before writing a GNU MathProg program to find the optimal solution for this problem, analyze it a little bit. You may recall from Part 2 in this series that the directional derivative of the objective function gives the direction in which the objective function grows in a maximization problem. Recall that the optimal solution is always on one of the vertices of the polyhedron created by the feasible region. When the directional derivative of the objective function is perpendicular to one of the sides of the polyhedron (formed by the constraints), all the points on that side of the polyhedron will have the same value for the objective function, because they lie on a same *iso-quanta*.

An iso-quanta is a line in space formed by points with the same objective function value. In this problem, $x_1+x_2=1$ is an iso-quanta, $x_1+x_2=2$ is another iso-quanta, and so on. If a constraint line is the farthest one along the directional derivative of the objective function and is perpendicular to that derivative, then all points on that boundary of the feasible region will make the objective solution reach its optimal value.

GNU MathProg solution for the multiple solution problem

The following is a GNU MathProg solution for the multiple solution problem:

Listing 4. Multiple solution problem solved with MathProg

```
 1  #
 2  # Multiple solution problem
 3  #
 4  # This finds the optimal solution for a simple multiple solution problem
 5  #
 6
 7  /* decision variables */
 8  var x1 >=0;
 9  var x2 >=0;
10
11  /* objective function */
```

```
12  maximize z: x1 + x2;
13
14  /* constraints */
15  s.t. Ctr: x1 + x2 <= 5;
16
17  end;
```

This very simple program declares the two decision variables: the constraint and the objective function.

Listing 5 shows the results:

Listing 5. glpsol's report

```
Problem:    multiple
Rows:       2
Columns:    2
Non-zeros:  4
Status:     OPTIMAL
Objective:  z = 5 (MAXimum)

   No.   Row name   St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------- ------------- ------------- -------------
     1 z            B              5
     2 Ctr          NU             5                           5             1

   No. Column name  St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------- ------------- ------------- -------------
     1 x1           B              5             0
     2 x2           NL             0             0                       < eps

Karush-Kuhn-Tucker optimality conditions:

KKT.PE: max.abs.err. = 0.00e+00 on row 0
        max.rel.err. = 0.00e+00 on row 0
        High quality

KKT.PB: max.abs.err. = 0.00e+00 on row 0
        max.rel.err. = 0.00e+00 on row 0
        High quality

KKT.DE: max.abs.err. = 0.00e+00 on column 0
        max.rel.err. = 0.00e+00 on column 0
        High quality

KKT.DB: max.abs.err. = 0.00e+00 on row 0
        max.rel.err. = 0.00e+00 on row 0
        High quality

End of output
```

The solution is optimal and the objective function value is 5. The second half of the report shows that the constraint is bounded and its marginal value is 1.

In the third section, the variable x_2 is on its lower bound and has a marginal value. This marginal value is reported as `<eps`, which is the unit of precision GLPK uses internally (in math, `epsilon` is commonly used to indicate a very small number, and this is the abbreviation). Something smaller than the minimal precision is very likely to be zero. Therefore, the x_2 variable would change the value of the objective function by 0 if it could ever be relaxed. In other words, a change in x_2 won't change the objective function's value, given that the constraint holds for the new (x_1, x_2) pair. So there are multiple points in the feasible region that yield the same value for the objective function. This problem

has multiple solutions!

Remember that whenever `glpsol` reports that the marginal value of a variable is `<eps`, that indicates a multiple solution problem.

Set covering (basketball) problem

The set covering problem teaches binary decision variables; that is, they can only be 0 or 1, yes or no.

> A basketball coach intends to set up his team for a game. The team consists of seven players from which five will play. The abilities of each player have been measured on a scale of 1 (poor) to 3 (excellent) for the skills of assisting, defending, throwing, and rebounding. There are three positions in which they can play: back-field (B), mid-field (M), and front-field (F). The positions in which each player can play and their abilities are shown in Table 1.

Table 1. Players and their abilities

Player	Position	Assisting	Throwing	Rebound	Defense
1	B	3	3	1	3
2	M	2	1	3	2
3	BM	2	3	2	2
4	MF	1	3	3	1
5	BF	1	3	1	2
6	MF	3	1	2	3
7	BF	3	2	2	1

The five chosen players must collectively satisfy the following constraints:

- At least three players must be capable of playing back-field.
- At least two players must be capable of playing front-field.
- At least one player must be capable of playing mid-field.
- The five players' average in terms of assisting, rebounding, defending, and throwing must be at least 2.
- If player 3 is playing, then player 6 can't play, and vice-versa.
- If player 1 is playing, then players 4 and 5 must also play.
- Either player 2 or player 3 must play.

The objective is to maximize the defensive ability of the players for this game.

Modeling the basketball problem

First, decide what decision variables to use. There are seven players on the team. Which five of them will play? Seven binary variables will decide if each player *i* will play (if the decision variable is 1) or not (0):

$$y_i \ binary, \ \forall i \in \{1, 2, ..., 6, 7\}$$

The objective function seeks to maximize the defensive ability of the players.

$$z = 3y_1 + 2y_2 + 2y_3 + y_4 + 2x_5 + 3y_6 + y_7$$

There is no need to divide the objective function by 5 to obtain the average, because maximizing `f(x)` and `f(x)` divided by `CONSTANT` is the same. In other words, the division doesn't change the direction that maximizes the objective function.

Next come the constraints. The first one says that at least three players must be able to play in the back-field. Only players 1, 3, 5, and 7 can do that:

$$y_1 + y_3 + y_5 + y_7 \geq 3 \ (Back)$$

This constraint, therefore, ensures that at least three of those four binary variables will be 1.

Similarly, for the mid-field, one of the players 2, 3, 4, and 6 is needed:

$$y_2 + y_3 + y_4 + y_6 \geq 1 \ (Mid)$$

The front-field needs two of the players 4, 5, 6, and 7.

$$y_4 + y_5 + y_6 + y_7 \geq 2 \ (Front)$$

The average ability to assist, rebound, defend, and throw must be at least 2:

$$\frac{3y_1 + 2y_2 + 2y_3 + y_4 + y_5 + 3y_6 + 3y_7}{5} \geq 2 \ (assist\ average)$$

$$\frac{3y_1 + y_2 + 3y_3 + 3y_4 + 3y_5 + y_6 + 2y_7}{5} \geq 2 \ (throw\ average)$$

$$\frac{y_1 + 3y_2 + 2y_3 + 3y_4 + y_5 + 2y_6 + 2y_7}{5} \geq 2 \ (rebound\ average)$$

$$\frac{3y_1 + 2y_2 + 2y_3 + y_4 + 2y_5 + 3y_6 + y_7}{5} \geq 2 \ (defense\ average)$$

The fifth constraint says that if player 3 plays, then player 6 can't play, and vice-versa. So only one of them can play:

$$x_3 + x_6 = 1$$

For this equation, if y_3=1, this constraint forces y_6 to be zero, and vice versa. This constraint forces either player 1 or 6 to be on the team for the game, though. That's not exactly what the coach wanted. They could both be out of the game occasionally, and this equation would not make it possible.

$$y_3 + y_6 \leq 1$$

Now, if y_3=1, y_6=0 because the sum can't be more than 1. If y_6=1, then y_3=0. If both y_3 and y_6 are zero, that's fine too.

The sixth constraint declares that when player 1 plays, players 4 and 5 must also play (perhaps it's in their contract). So, y_4 and y_5 must be 1 when $y_1=1$:

$$y_4 \geq y_1$$

$$y_5 \geq y_1$$

This pair of inequalities is tricky. If $y_1=0$, $y_4 >= 0$. This means that player 4 may play if player 1 is not playing. If player 1 plays, however, $y_4 >= 1$, so y_4 equals 1. When player 1 is playing, player 4 is also playing. The same type of inequality is enforced for player 5.

The last constraint says that either players 2 or 3 must play. Note that they can't both play at the same time:

$$y_2 + y_3 = 1$$

Remember the first try for the constraint regarding players 3 and 6? This constraint enforces that either player 2 or 3 will be on the game, but not both.

There's a hidden constraint: a basketball team has five players on the court. Therefore:

$$\sum_{i=1}^{7} y_i = 5 \ \ (five\ players)$$

To ensure that the decision variables are binary, they are declared to take only values in a binary set:

$$y_i \in \{0,1\}, \forall i$$

Because this problem is actually the selection of decision variables to cover some constraints, it's called a *set covering* problem.

GNU MathProg solution for the set covering problem

As above, the line numbers in this code are not part of the code itself. They have been added only for the sake of making references to the code later.

Listing 6. GNU MathProg solution to the set covering problem

```
 1      #
 2      # Basketball problem
 3      #
 4      # This finds the optimal solution for maximizing the team's overall
 5      # defensive skill
 6      #
 7
 8      /* sets */
 9      set SKILLS;
10      set POSITIONS;
```

```
11      set PLAYERS;
12
13      /* parameters */
14      param skill {i in PLAYERS, j in SKILLS};
15      param position {i in PLAYERS, j in POSITIONS};
16      param min_in_position {i in POSITIONS};
17      param min_skill_average {i in SKILLS};
18
19      /* decision variables: yi, i in {1,..,7}. yi = 1 -> player i is on team */
20      var y {i in PLAYERS} binary >=0;
21
22      /* objective function */
23      maximize z: sum{i in PLAYERS} skill[i,"defense"]*y[i];
24
25      /* Constraints */
26      s.t. pos{j in POSITIONS}: sum{i in PLAYERS}
              position[i,j]*y[i] >= min_in_position[j];
27      s.t. players: sum{i in PLAYERS} y[i] = 5;
28      s.t. ctr3: y[3] + y[6] <= 1;
29      s.t. ctr4a: y[4] - y[1] >= 0;
30      s.t. ctr4b: y[5] - y[1] >= 0;
31      s.t. ctr5: y[2] + y[3] = 1;
32      s.t. average{j in SKILLS}: sum{i in PLAYERS}
              skill[i,j]*y[i]/5 >= min_skill_average[j];
33
34      data;
35      set PLAYERS := 1 2 3 4 5 6 7;
36      set POSITIONS := Back Mid Front;
37      set SKILLS := assist throw rebound defense;
38
39      param min_in_position:=
40      Back    3
41      Mid     1
42      Front   2;
43
44      param min_skill_average:=
45      assist          2
46      throw           2
47      rebound         2
48      defense         2;
49
50      param position: Back    Mid     Front:=
51      1               1       0       0
52      2               0       1       0
53      3               1       1       0
54      4               1       0       1
55      5               1       0       1
56      6               0       1       1
57      7               1       0       1;
58
59
60      param skill:    assist  throw   rebound defense:=
61      1               3       3       1       3
62      2               2       1       3       2
63      3               2       3       2       2
64      4               1       3       3       1
65      5               1       3       1       2
66      6               3       1       2       3
67      7               3       2       2       1;
68
69      end;
```

There are four parameters:

- `skill` is a two-dimensional table where `i` denotes the player and `j` lists the skills. The values of

this table are the numerical ability of player `i` with skill `j`.

- `position` is a two-dimensional binary table with players along `i` and `j` listing the positions. This table determines whether player `i` can play on position `j` (if the value is 1).
- `min_in_position` is defined on the `POSITIONS` set to define the minimum number of players that can play position `i`.
- `min_skill_average` is defined on the `SKILLS` set to set the minimum average the playing players must have regarding skill `i`. This average happens to be the same for all of them as given in the problem, but it should be declared separately in case the coach needs to shift the team's balance.

The heart of this problem is the declaration of the decision variables as binary variables. They are as easy to declare as integer variables. Just add a `binary` to the declaration as shown on line 20.

Here's the output:

Listing 7. glpsol report for the set covering problem

```
Problem:    basketball
Rows:       13
Columns:    7 (7 integer, 7 binary)
Non-zeros:  62
Status:     INTEGER OPTIMAL
Objective:  z = 11 (MAXimum) 11 (LP)

   No.   Row name        Activity     Lower bound   Upper bound
------ ------------    ------------- ------------- -------------
     1 z                          11
     2 pos[Back]                   3             3
     3 pos[Mid]                    2             1
     4 pos[Front]                  3             2
     5 players                     5             5             =
     6 ctr3                        1                           1
     7 ctr4a                       0            -0
     8 ctr4b                       0            -0
     9 ctr5                        1             1             =
    10 average[assist]
                                   2             2
    11 average[throw]
                                 2.2             2
    12 average[rebound]
                                   2             2
    13 average[defense]
                                 2.2             2

   No. Column name       Activity     Lower bound   Upper bound
------ ------------    ------------- ------------- -------------
     1 y[1]         *              1             0             1
     2 y[2]         *              1             0             1
     3 y[3]         *              0             0             1
     4 y[4]         *              1             0             1
     5 y[5]         *              1             0             1
     6 y[6]         *              1             0             1
     7 y[7]         *              0             0             1

End of output
```

Note that all the decision variables are either 0 or 1, as expected.

Conclusion

This series of articles analyzed six problems, each one teaching a new concept through modeling the problem, writing it in a simple and elegant way in the GNU MathProg language so that GLPK can solve it, and analyzing the results.

The uses of linear programming and Operations Research to search for optimal solutions do not end with the problems in this series of articles. Areas such as petroleum, chemistry, food, and finance use linear solvers and Operations Research heavily.

I encourage you to use and share the concepts discussed here, so that more people learn the power of linear programming and Operations Research. I hope you found the material useful.

Resources

Learn

- Read the entire series, "The GNU Linear Programming Kit" (developerWorks, August and September 2006).
- The problems in this article are taken with permission from *Operations Research: Applications and Algorithms, 4th Edition*, by Wayne L. Winston (Thomson, 2004).
- The online documentation for GLPK gives more information about GLPK, how to get the software, and how to join the GLPK community.
- Subscribe to the GLPK help mailing list or bug reports mailing list.
- In the developerWorks Linux zone, find more resources for Linux developers.
- Stay current with developerWorks technical events and Webcasts.

Get products and technologies

- With IBM trial software, available for download directly from developerWorks, build your next development project on Linux.

Discuss

- Check out developerWorks blogs and get involved in the developerWorks community.

About the author

Rodrigo Ceron Ferreira de Castro is a Staff Software Engineer at the IBM Linux Technology Center. He graduated from the State University of Campinas (UNICAMP) in 2004. He received the State Engineering Institute prize and the Engineering Council Certification of Honor when he graduated. He's given speeches in open source conferences in Brazil and other countries.